IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPELLANT:	CHRISTOPHER A. KRYGOWSKI ET AL.	) Group Art Unit: 2124
SERIAL NO.:	09/764,595	) Examiner: DO, CHAT C
FILED:	January 18, 2001	)
FOR:	FLOATING POINT UNIT FOR MULTIPLE DATA ARCHITECTURES	) Confirmation No. 6652

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

1. THE REAL PARTY IN INTEREST

The real party in interest in this appeal is International Business Machines Corporation. Ownership by International Business Machines Corporation is established by an assignment document recorded for this application on May 24, 2001 on Reel 011850 Frame 0940.

2. RELATED APPEALS AND INTERFERENCES

Appellants know of no related patent applications or patents under any appeal or interference proceeding.

3. STATUS OF CLAIMS

Currently, claims 1-12 and 14-18 are pending. All pending claims stand rejected under 35 U.S.C. § 103(a).

4. STATUS OF AMENDMENTS

There have been no amendments filed subsequent to receipt of the final office action.

5. SUMMARY OF INVENTION

The following is a concise explanation of the invention. Reference to the specification and drawings is made pursuant to 37 CFR 1.192 and is not intended to limit the claims to the embodiments shown and described in the application.

Referring to Figure 1 of the application, the invention relates to a computer system having a floating point unit for supporting multiple floating point architecture input types. In order to understand the invention, some background information may be helpful. Software applications that are designed for one floating point architecture (e.g., binary, hexadecimal) may not be compatible with another floating point architecture. However, it is often desirable to run the same application on different hardware platforms supporting different floating point architectures, or to use data generated by an application on one platform with a different application on another platform that supports a different floating point architecture. Moreover, it is often desirable to run applications that are based on different floating point architectures on a single machine. These requirements are particularly relevant to multi-tasking environments where task switching between different applications may require using different floating point architectures.

It is desirable to create a floating point unit for a computer system that supports multiple floating point architectures. Some previous attempts converted

all binary operands into a hexadecimal-like format and operated internally in just one format. However, performance was sacrificed in these implementations because two additional cycles were required to format binary into hexadecimal, and then to reformat a hexadecimal result into binary. Other previous attempts optimized the data-flow for binary, and converted only the hexadecimal operands into the binary format. However, this penalized hexadecimal operands with format conversion cycles.

The invention provides a computer system with a floating point unit for supporting multiple floating point architectures without incurring additional clock cycles for the conversion between the floating point architectures and an internal floating point format. Floating point architectures supported by the present invention include a binary architecture and a hexadecimal architecture. A shared internal binary format is utilized with two different biases to simplify the conversion process for both binary external architectures and hexadecimal external architectures. The format conversion process to and from the internal binary format does not adversely impact performance.

A typical floating point number includes three components: an exponent, a fraction and a sign bit. In embodiments of the invention, the exponent format conversion for a binary external architecture is performed by a multiplexer and is given by the following expression: Internal Exponent(0) = Architected Exponent(0); Internal Exponent(1:15 - (n -1)) = (NOT Architected Exponent(0)) || (NOT Architected(0)) ||... || (NOT Architected Exponent(0)); and Internal Exponent(15 - (n -1):15) = Architected Exponent(1:n), where n is the length of the architected exponent format. The exponent conversion for a hexadecimal external architecture is performed by a multiplexer and is given by the following expression: Internal Exponent(0) = Architected Exponent(0); Internal Exponent(1:7) = (NOT Architected Exponent(0)) || (NOT Architected(0)) ||... || (NOT Architected Exponent(0)); and Internal Exponent(8:15) = Architected

Exponent(1:6) || '00'. The sign bits and fractions do not require any conversions and are equivalent in both the shared internal binary format and the external architectures.

Because no calculations are required to perform a conversion from an external architecture to an internal format, the format conversion requires no additional delay, or cycles, and can be performed by multiplexing the data as the data is being input. The format conversion takes place via a multiplexer as the data is received into the input data register 16 of the floating point execution unit 10 as shown in Figure 1. The conversion from internal binary format to an external architecture is simply the reverse of the process that was performed to convert to the internal format. This process is achieved by controlling the result register input multiplexer.

6. ISSUE

There is one issue on appeal: whether the Examiner's rejection of claims 1-12 and 14-18 under 35 U.S.C. § 103(a) as being anticipated by Schwarz et al. (U.S. 5,687,106) ("Schwarz") is improper.

7. GROUPING OF CLAIMS

There are two groups of claims. Claims 1, 3-6, 10-12 and 14-18 comprise the first group, which stand or fall together, under the Examiner's contested rejection of these claims under 35 U.S.C. § 103(a) as being anticipated by Schwarz. Claims 2 and 7-9 comprise the second group, under the Examiner's contested rejection of this claim under 35 U.S.C. § 103(a) as being anticipated by Schwarz.

8. ARGUMENT

A. Summary of Schwarz

Prior to addressing specific claims a summary of the Schwarz patent is provided. Schwarz is directed to a computer system supporting multiple floating point architectures. A hexadecimal floating point architecture and a binary floating point architecture are supported by a hexadecimal internal dataflow which accommodates both formats. All operations are executed in the hexadecimal internal data flow. The system includes format conversion means between the hexadecimal internal dataflow and the architected data types. (See Col. 1 lines 57-65.) The conversion from hexadecimal architected to hexadecimal internal is relatively straightforward given the representation of hexadecimal architected (56 fraction bits, 7 exponent bits, 1 sign bit, and a hexadecimal exponent bias of 64) to hexadecimal internal notation (56 fraction bits, 14 exponent bits, 1 sign bit, and a hexadecimal exponent bias of 8192). Schwarz teaches that no extra stages are required for the conversion from hexadecimal architected to hexadecimal internal. (See Col. 4 lines 4-15.)

The conversion process from binary architected to hexadecimal internal as taught by Schwarz is more complex than the conversion from hexadecimal architected to hexadecimal internal. Schwarz teaches that the conversion from binary architected to hexadecimal internal includes two portions, a fraction conversion portion to perform fraction alignment to the hexadecimal boundary and an exponent conversion portion that includes shifting and incrementing of the exponent. (See Fig. 2 and Fig. 3) The conversion back to binary architected from hexadecimal internal includes similar processing. Schwarz teaches providing a floating point unit having very high performance on hexadecimal operations and providing compatibility of binary operations, without significant loss of

performance for these binary operations. (See Col. 20 lines 43-46.)

B. Claims 1, 3-6, 10-12 and 14-18 are patentable over Schwarz under 35 U.S.C. § 103(a).

Under the first grouping of claims, the Examiner improperly rejected Claims 1, 3-6, 10-12 and 14-18 under 35 U.S.C. § 103(a) as being unpatentable over Schwarz. For an obviousness rejection to be proper, the Examiner must meet the burden of establishing that all elements of the invention are disclosed in the prior art; and that the prior art relied upon, coupled with knowledge generally available in the art at the time of the invention, must contain some suggestion or incentive that would have motivated the skilled artisan to modify a reference or combined references. *In re Fine*, 5 U.S.P.Q.2d 1596, 1598 (Fed. Cir. 1988); *In Re Wilson*, 165 U.S.P.Q. 494, 496 (C.C.P.A. 1970); *Amgen v. Chugai Pharmaceuticals Co.*, 927 U.S.P.Q.2d, 1016, 1023 (Fed. Cir. 1996).

Schwarz does not teach or suggest all of the limitations set forth in Claims 1, 3-6, 10-12 and 14-18. Claims 1, 3-6, 10-11 and 18 include the following element: "wherein said converting into the internal floating point format occurs without incurring additional clock cycles when said one of the plurality of floating point architectures is binary." Claims 12 and 14-17 include a similar element.

The Examiner states that Schwarz does not implicitly disclose the converting into the internal floating point format without incurring additional clock cycles when one of the plurality of floating point architectures is binary. However, the Examiner asserts that Schwarz does not disclose a requirement for an additional clock cycle to convert from a binary floating point architecture to an internal floating point format. Please note that Appellants are assuming, based on the context, that the Examiner meant to say that "Schwarz et al. neither implicitly disclose the converting into the internal floating point format occurs **with** incurring additional clock cycles when one of the plurality of floating point

architecture is binary ..." at page 3, lines 14-16 in the final office action. The Examiner further asserts that it is well known in the art to operate multiple data operations in a single instruction and that it would have been obvious to add the step of converting into the internal floating point format without incurring additional clock cycles in order to increase system performance. Appellants strongly disagree with the Examiner's assertions.

Column 4, lines 42-47 in Schwartz, as referred to by the Examiner, describe receiving input data, and in the next cycle transmitting the data to a hexadecimal internal dataflow 18 if the input data is in hexadecimal format. Alternatively, the input data is transmitted to a binary architected to hexadecimal internal converter 15 if the input data is in binary format. After the binary input data is converted to a hexadecimal internal format, it is transmitted to the hexadecimal internal dataflow 18 **in the following cycle**. This extra cycle is depicted in Figure 1 by the loop from the A register 13 to the binary architected to hexadecimal converter 15 and back into the A register 13. A similar cycle is depicted from the B register 14. This section of Schwarz clearly teaches that the conversion from binary architected to hexadecimal internal dataflow 18 requires an extra cycle that is not required when the input data is in hexadecimal format.

The Examiner also refers to Figure 5 to support his assertion that Schwarz does not disclose a requirement for an additional clock cycle to convert from a binary floating point architecture to an internal floating point format. Appellants respectfully submit that Figure 5 and the accompanying text teach an additional clock cycle to convert from a binary input architecture to the internal format. Schwarz recites "the binary data must be converted into hex data ... the first cycle of execution involves format conversion" at Column 18 line 66 through Column 19 line 1. Schwarz teaches that the binary architected to hexadecimal internal conversion includes converting both the fraction portion and the exponent portion of the binary architected input. Figure 2 in Schwarz depicts a fraction conversion

macro that includes performing a relatively complicated fraction alignment to the hexadecimal boundary operation. Figure 3 in Schwarz depicts an exponent conversion macro that includes shifting and incrementing the exponent as well as detecting special cases of exponents. Because of the amount of processing required and the dependencies between the processing steps, the processing depicted in Figures 2 and 3 in Schwarz requires at least one additional clock cycle to convert the binary architected input data into the hexadecimal internal format. As discussed previously, this extra clock cycle is depicted in FIG. 1 by the loop from the A register 13 to the binary architected to hexadecimal converter 15 back into the A register 13.

The Examiner has also asserted that it would have been obvious to a person having ordinary skill in the art to add a step of converting into the internal floating point format without incurring additional clock cycles to increase the system performance. Appellants respectfully traverse the Examiners assertion. It is not obvious how to modify the complicated format conversion process described in Schwarz (see Figures 2 and 3 and the accompanying text) to result in a conversion process that can be performed in the same clock cycle as the data input.

In the Appellants claimed invention, an extra clock cycle is not required to convert from a binary floating point architecture to the internal floating point format. Nor is an extra clock cycle required to convert from a hexadecimal floating point architecture to the internal floating point format.

Thus, because Schwarz does not teach or suggest "said converting into the internal floating point format occurs without incurring additional clock cycles when said one of the plurality of floating point architectures is binary", Appellants respectfully submit that the rejection of Claims 1, 3-6, 10-12 and 14-18 under 35 U.S.C. § 103(a) is improper.

C. Claims 2 and 7-9 are patentable over Schwarz under 35 U.S.C. § 103(a).

Under the second grouping of claims, the Examiner improperly rejected Claim 2 and 7-9 under 35 U.S.C. § 103(a) as being unpatentable over Schwarz.

Schwarz does not teach or suggest all of the limitations set forth in Claim 2. Claim 2 includes the following element: "the fraction bits corresponding to each of the plurality of floating point architectures are used by the floating point unit in an unconverted state" and Claims 7-9 include a similar element.

The Examiner asserts that Schwarz teaches "the fraction bits corresponding to each of a plurality of floating point architectures are used by the floating point unit in an unconverted state (table 1)." Appellant disagrees with the Examiner's assertions.

Figure 2 and the accompanying text in Columns 7 and 8 of Schwarz describe the fraction portion of binary architected to hexadecimal internal converters. The fraction conversion is separated into two functions: the fraction conversion and the detection of special numbers. Schwarz does not teach or suggest that "the fraction bits ... are used by the floating point unit in an unconverted state" as recited in Appellants' Claim 2. Indeed, Schwarz actually teaches away from "the fraction bits ... are used by the floating point unit in an unconverted state" because Schwarz specifically teaches how to convert the fraction portion of a binary architected input into the fraction portion of a hexadecimal internal format.

Thus, because Schwarz does not teach or suggest "the fraction bits ... are used by the floating point unit in an unconverted state" and it is not obvious from the disclosure of Schwarz that "the fraction bits ... are used by the floating point unit in an unconverted state", Appellants respectfully submit that the rejection of claims 2 and 7-9 under 35 U.S.C. § 103(a) is improper.

J. Conclusion

For the reasons cited above, Appellants respectfully submit that the rejections are improper and request reversal of the outstanding rejections. If there are any additional charges with respect to this Appeal, or otherwise, please charge them to Deposit Account No. 09-0463 maintained by Appellants' Assignee.

Respectfully submitted,

CANTOR COLBURN LLP

By: _____

Anne Davis Barry
Registration No. 47, 408
CANTOR COLBURN LLP
55 Griffin Road South
Bloomfield, CT 06002
Telephone (860) 286-2929
Facsimile (860) 286-0115
Customer No. 23413

April 27, 2004

APPENDIX A

Appealed Claims

1. A computer system for supporting a plurality of floating point architectures, each floating point architecture having at least one format, the system comprising:

a floating point unit having an internal data-flow according to an internal floating point format for performing floating point operations in the internal format, wherein the internal format has a number of exponent bits which is at least a minimum number required to support each of the plurality of floating point architectures and the internal format has a number of fraction bits which is at least the minimum number required to support each of the plurality of floating point architectures; and

a converter for converting an exponent value corresponding to each one of the plurality of floating point architectures into the internal floating point format such that an operand of any one of the plurality of floating point architectures input to the floating point unit is converted into the internal floating point format for operation by the floating point unit, and the result of the operation is converted back into the one of the plurality of floating point architectures by converting an exponent value corresponding to the internal floating point format into the one of the plurality of floating point architectures, wherein said converting into the internal floating point format occurs without incurring additional clock cycles when said one of the plurality of floating point architectures is binary and said converting into the internal floating point format occurs without incurring additional clock cycles when said one of the plurality of floating point architectures is hexadecimal.

2. The computer system of claim 1, wherein the fraction bits corresponding to each of the plurality of floating point architectures are used by the floating point unit in an unconverted state.

3. The computer system of claim 1, wherein the plurality of floating point architectures include IBM®-S/390® hexadecimal floating point architecture and IEEE-754 binary floating point architecture.

4. The computer system of claim 3, wherein the converter:

determines a sign bit; and

normalizes a resulting binary floating point number according to at least one of IBM®-S/390® and IEEE-754 normalization modes.

5. The computer system of claim 3, wherein the internal floating point format is a binary format that has a 16 bit exponent biased by 32,768, a sign bit, and a 56 bit fraction.

6. The computer system of claim 1, wherein the plurality of floating point architectures includes a binary architected format and a hexadecimal architected format, and the internal format is a binary internal format.

7. The computer system of claim 6, wherein the binary internal format has a common predetermined fraction type corresponding to both of the hexadecimal and the binary architected formats.

8. The computer system of claim 7, wherein the numbers represented in the binary internal format corresponding to each of the hexadecimal architected format and the binary architected format, respectively, each have predetermined

bias types that differ in the locations of the implied radix points.

9. The computer system of claim 8, wherein the binary internal format has a nonzero positive integer number M of exponent bits and a bias equal to $2^{(M-1)} - 1$, where M is the length of the internal exponent field.

10. The computer system of claim 1, wherein the plurality of floating point architectures includes a binary architected format and a hexadecimal architected format, the internal format is a binary internal format, and the converter comprises:

a first converter portion that converts the hexadecimal architected format to the binary internal format, and converts the binary architected format to the binary internal format; and

a second converter portion that converts the binary internal format into the binary architected format, and converts the binary internal format into the hexadecimal architected format.

11. The computer system of claim 10, wherein the first converter portion comprises an input format conversion multiplexor and input format conversion control, and the second converter portion comprises an output format conversion multiplexor and output format conversion control.

12. A floating point unit for supporting a first floating point architecture and a second floating point architecture, the first and second floating point architectures each having at least one format, the unit comprising:

an internal floating point format compatible with both the first and second floating point architectures, and sharing the fraction bits and sign bit with both of the floating point architectures;

a first converter to convert an operand of the first floating point architecture type to the internal floating point format by multiplexing the exponent bits of the operand, and to convert an operand of the second floating point architecture type to the internal floating point format by multiplexing the exponent bits of the operand, wherein said first converter operates without incurring additional clock cycles when said first floating point architecture type is binary, said first converter operates without incurring additional clock cycles when said first floating point architecture type is hexadecimal, said first converter operates without incurring additional clock cycles when said second floating point architecture type is binary and said first converter operates without incurring additional clock cycles when said second floating point architecture type is hexadecimal;

a floating point unit having an internal data-flow for the internal floating point format that supports the converted data of both the first and second floating point architectures, and that performs floating point operations on data formatted in the internal floating point format by the first converter; and

a second converter to convert data of the internal floating point format into data of the first floating point architecture by multiplexing the exponent bits of the operand, and to convert data of the internal floating point format into data of the second floating point architecture by multiplexing the exponent bits of the operand.

14. The floating point unit of claim 12, wherein the data in the internal floating point format is directly converted into data of the first floating point architecture by the second converter with no additional delay, and wherein data in the internal floating point format is directly converted into data of the second floating point architecture by the second converter with no additional delay.

15. The floating point unit of claim 12, wherein the internal floating point format has a minimum nonzero positive number of exponent bits, N, to support both first and second floating point architectures.

16. The floating point unit of claim 12, wherein the internal floating point format has a single fraction type corresponding to the fraction portions of both the first and second floating point architectures.

17. The floating point unit of claim 12, wherein the internal format has the same fraction type as both of the first and second floating point architectures.

18. A method for processing data corresponding to a plurality of floating point architectures, comprising:

determining a type of a floating point architecture;

converting the exponent of the data corresponding to the determined architecture into an internal floating point format by:

correcting the radix point location of the exponent data; and

pre-aligning the corrected exponent data, wherein said converting is performed without incurring additional clock cycles when said determined architecture is binary and said converting is performed without incurring additional clock cycles when said determined architecture is hexadecimal;

performing arithmetic computations on the converted data;

re-converting the exponent of the computed data by:

re-correcting the radix point location of the exponent of the computed data, and

post-aligning the re-corrected exponent of the computed data; and

outputting the re-converted data to thereby provide floating point data corresponding to the original floating point architecture without incurring a

delay beyond that required to compute the resulting output in the internal floating point format.